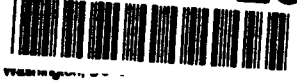
(2)

AD-A236 269



TATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final: 15 Aug 1990 to 01 Mar 1991

4. TITLE AND SUBTITLE
York Software Engineering Limited, York Ada Compiler Envionment (ACE) Release 5, Intergraph InterPro 3050 Workstation (Host & Target), 901127N1.11073

5. FUNDING NUMBERS

6. AUTHOR(S)
National Computing Centre Limited
Manchester, UNITED KINGDOM

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Computing Centre Limited
Oxford Road
Manchester MI 7ED
UNITED KINGDOM

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF_VSR_90502/70-910415

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
York Software Engineering Limited, York Ada Compiler Environment (ACE) Release 5, Manchester England, Intergraph InterPro 3050 Workstation (under CLIX R3.1)(Host & Target), ACVC 1.11.

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on **901127.**

Compiler Name and Version:	**York Ada Compiler Environment (ACE) Release 5**
Host Computer System:	**Intergraph InterPro 3050 Workstation (under CLIX R3.1)**
Target Computer System:	**Intergraph InterPro 3050 Workstation (under CLIX R3.1)**

A more detailed description of this Ada implementation is found in section 3.1 of this report.
As a result of this validation effort, Validation Certificate **#901127N1.11073** is awarded to **York Software Engineering Limited.** This certificate expires on **01 JUNE 1992.**

This report has been reviewed and is approved.

A.E.J. Pink

Jane Pink
Testing Services Manager
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
England

Ada Validation Organization
for Director, Computer & Software
Engineering Division
Institute for Defense Analyses
Alexandria
VA 22311

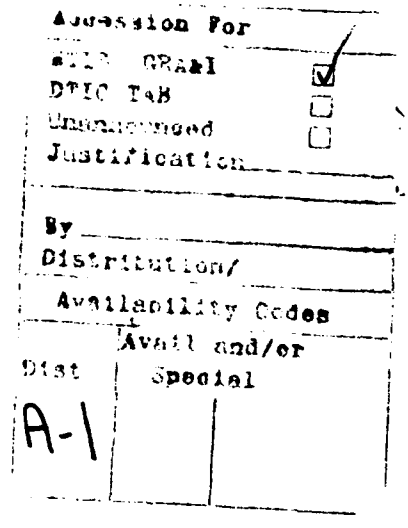
William Lewis Collett

for Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington
DC 20301

AVF Control Number: AVF_VSR_90502/70-910415

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: #901127N1.11073
York Software Engineering Limited
York Ada Compiler Environment (ACE) Release 5
Intergraph InterPro 3050 Workstation

Prepared by
Testing Services
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
England



VSR Version 90-08-15

The following declaration of conformance was supplied by the customer.

DECLARATION OF CONFORMANCE

Customer: **York Software Engineering Limited**

Ada Validation Facility: **The National Computing Centre Limited**
Oxford Road
Manchester
M1 7ED
United Kingdom

ACVC Version: **1.11**

Ada Implementation:

Ada Compiler Name: **York Ada Compiler Environment (ACE)**

Version: **Release 5**

Host Computer System: **Intergraph InterPro 3050 Workstation (under CLIX R 3.1)**

Target Computer System: **Intergraph InterPro 3050 Workstation (under CLIX R 3.1)**

Customer's Declaration

I, the undersigned, representing **York Software Engineering Limited**, declare that **York Software Engineering Limited** has no knowledge of deliberate deviations from the Ada Language Standard **ANSI/MIL-STD-1815A** in the implementation(s) listed in this declaration.

Signature

27 - 11 - 1990
Date

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield
VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria
VA 22311

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide,
21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behaviour is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.
For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 3.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.

Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada Certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several interconnected units.
Conformity	Fulfilment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.

Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 90-10-12.

E28005C	B28006C	C34006D	B41308B	C43004A	C45114A
C45346A	C45612B	C45651A	C46022A	B49008A	A74006A
C74308A	B83022B	B83022H	B83025B	B83025D	B83026B
B85001L	C83026A	C83041A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
BD8002A	BD8004C	CD9005A	CD9005B	CDA201E	CE2107I
CE2117A	CE2117B	CE2119B	CE2205B	CE2405A	CE3111C
CE3118A	CE3411B	CE3412B	CE3607B	CE3607C	CE3607D
CE3812A	CE3814A	CE3902B			

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. The inapplicability criteria for some tests are explained in documents issued by ISO and the AJPO known as Ada Issues and commonly referenced in the format AI-dddd. For this implementation, the following tests were inapplicable for the reasons indicated; references to Ada Issues are included as appropriate.

B23003D and B23003E check that the limit of the input-line length is correctly enforced; this implementation does no such limit. (See section 2.3).

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

In addition, the three tests C24113W..Y, included above, contain literals that exceed the implementation's maximum numeric literal length of 256.

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35702A, C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C45624A checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 5, For this implementation, MACHINE_OVERFLOWS is TRUE.

C45624B checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 6. For this implementation, 'MACHINE_OVERFLOWS is TRUE.

C4A013B contains the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type. This expression would raise an exception. Since the expression must be static, it is rejected at compile time.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CA2009C, and CA2009F, compile generic specifications and bodies or specifications and bodies of subunits of generic units in separate files. This compiler requires that no instantiations of the corresponding generics occur prior to the compilations of the generic body.

LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F check for pragma INLINE for procedures and functions.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

BD9001A, AD9001B, ED9002A, AD9004A and BD9004B use pragma INTERFACE.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

The tests listed in the following table are not applicable because the given file operations are supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	--------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

The following 15 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file; USE_ERROR is raised when this association is attempted.

CE2107A..B	CE2107E..G	CD2110B	CE2110D	CE2111D
CE2111H	CE3111A..B	CE3111D..E	CE3114B	CE3115A

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

CE2403A checks that WRITE raises USE-ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3304A checks that USE_ERROR is raised if a call to SET_LINE_LENGTH or SET_PAGE_LENGTH specifies a value that is inappropriate for the external file. This implementation does not have inappropriate values for either line length or page length.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 64 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada standard in the way expected by the original tests.

B22003A	B28003A	B28003C	B29001A
B2A003A	B2A007A	B2A010A	B32103A
B35103B	B37301B	B43201A	B44001A
B44004A	B49007A	B54A20A	B71001C
B83011A	B83011B	B83012A	B83E01A
B83E01B	B91001H	B95004B	B95061G
B95077A	B97102H	BA3013A	BA1101B
BA1101C	BA3006A	BA3006B	BA3007B
BA3008A	BA3008B	BC1001A	BC1016A
BC1202E	BC2001E	BC2061D	BD2B14A
BD2D03A	BD4003A	BE2201A	BE2413A

The following tests were split to remove any spurious errors reported after a legitimate and expected error report.

B23004A	B33102A	B38101C	B51001A
B54A01C	B55A01A	B55B17A	B66001C
B91002F	B95001D	BC3204B	BC3204D
BC3205D	BD2B03A	BD2C14A	BD3003B

B23003D and B23003E were graded inapplicable by Evaluation Modification as directed by the AVO. These tests check that an implementation imposes a limit on the length of the input line; this implementation has no such limit. The AVO ruled that this behaviour is acceptable.

B23003F was graded passed by Evaluation Modification as directed by the AVO. This test checks that an identifier may not exceed the limit on the input line length. Although this implementation imposes no such limit, it does limit identifiers to 128 characters (and numeric literals to 256); the AVO ruled that this behaviour is acceptable, and that this test thus constitutes a check that the identifier limit is correctly enforced.

BC3204C and BC3205D when processed produced no errors because the compiler requires that no instantiations of the corresponding generics occur prior to the completion of the generic body. The tests were passed by applying the Processing Modification of re-compiling the obsolete main program, which caused all the illegal instantiations to be detected. The processing modification resulted in a split being required to BC3205D to prove that legal instantiations do not produce errors (See above list).

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested with ACVC 1.11 was the York Ada Compiler Environment (ACE) Release 5, running on an Intergraph InterPro 3050 Workstation with 16mb main memory. The Operating System was CLIX R3.1.

For a point of contact for technical and sales information, see:

Ms Moira West
York Software Engineering Limited
University of York
York
YO1 5DD

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

a)	Total Number of Applicable Tests	3769	
b)	Total Number of Withdrawn Tests	81	
c)	Processed Inapplicable Tests	320	
d)	Non-Processed I/O Tests	0	
e)	Non-Processed Floating-Point Precision Tests	0	
f)	Total Number of Inapplicable Tests	320	(c+d+e)
g)	Total Number of Tests for ACVC 1.11	4170	(a+b+f)

All I/O tests of the test suite were processed because this implementation supports a file system. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 320 tests were

inapplicable to this implementation. All inapplicable tests were processed during validation testing. In addition, the modified tests mentioned in section 2.3 were also processed.

A **Data Cartridge Tape** containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the **Data Cartridge Tape** were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

Compiler Options:

-M "identifier"
Make the compilation unit "identifier" the main subprogram.

-c
Suppress the loading of resulting object files. This is used when compiling individual subprogram units when other units that are needed to produce an executable file may not yet have been compiled.

-P
Generate a compilation listing. For a source file with a "A" suffix a compilation listing is produced in a file with a ".lst" suffix.

Test output, compiler and linker listings, and job logs were captured on **Data Cartridge Tape** and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the valued that are defined in terms of the maximum input-line length, which is the value for $MAX_IN-LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$MAX_IN_LEN	128
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

MACRO PARAMETERS

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	65536
$DEFAULT_MEM_SIZE	2147483647
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	UNIX
$DELTA_DOC	0.000000000465661287307739257812 5
$ENTRY_ADDRESS	SPARE_MEM'ADDRESS
$ENTRY_ADDRESS1	SPARE_MEM'ADDRESS
$ENTRY_ADDRESS2	SPARE_MEM'ADDRESS
$FIELD_LAST	80
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_FIXED
$FLOAT_NAME	NO_FLOAT
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	16777200.0
$GREATER_THAN_DURATION_BASE_LAST	2147483648.0
$GREATER_THAN_FLOAT_BASE_LAST	3.402_83E+38
$GREATER_THAN_FLOAT_SAFE_LARGE	1.701_42E+38
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	1.0E308

$HIGH_PRIORITY	4
$ILLEGAL_EXTERNAL_FILE_NAME1	MUCH_TOO_LONG_NAME_FOR_A_FILE
$ILLEGAL_EXTERNAL_FILE_NAME2	MUCH_TOO_LONG_NAME_FOR_A_FILE_1
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INCLUDE_PRAGMA1	PRAGMA INCLUDE ("A28006D1.TST")
$INCLUDE_PRAGMA2	PRAGMA INCLUDE ("B28006D1.TST")
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$INTEGER_LAST_PLUS_1	2147483648
$INTERFACE_LANGUAGE	C
$LESS_THAN_DURATION	-16777200.0
$LESS_THAN_DURATION_BASE_FIRST	-2147483648.0
$LINE_TERMINATOR	ASCII.LF
$LOW_PRIORITY	0
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	NO_SUCH_TYPE
$MANTISSA_DOC	31
$MAX_DIGITS	15
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2147483648
$MIN_INT	-2147483648
$NAME	BYTE_INTEGER
$NAME_LIST	UNIX

$NAME_SPECIFICATION1	X2120A
$NAME_SPECIFICATION2	X2120B
$NAME_SPECIFICATION3	X3119A
$NEG_BASED_INT	16#FFFFFFFD#
$NEW_MEM_SIZE	2147483647
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	UNIX
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	NEW_INTEGER;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.01
$VARIABLE_ADDRESS	SPARE_MEM'ADDRESS
$VARIABLE_ADDRESS1	SPARE_MEM'ADDRESS
$VARIABLE_ADDRESS2	SPARE_MEM'ADDRESS
$YOUR_PRAGMA	LIBNAME

APPENDIX B

COMPILATION SYSTEM OPTIONS

The following compiler options are available

-M *identifier*

Make the compilation unit *identifier* the main subprogram. The main subprogram must be a parameterless procedure body. "-M main" is assumed if this option is not specified.

-v Have the compiler produce verbose error messages. In particular, the compiler will attempt to isolate faults within expressions detected during overload resolution.

-w Suppress all warning messages from the Ada compiler.

-c Do not load the resulting .o files together.

-S Save the assembly code output of the compiler in a file with suffix ".s" or ".S". No object files will be produced.

-o *outfile*

Name the resultant file *outfile* rather than the default "a.out".

-O Invoke the object code improver on the compiler output.

-P Generate compilation listings. A compilation listing is produced in a file with a ".lst" or ".LST" suffix for each Ada source file on the command line.

-g Generate extra symbol table information for *Adb(I)*.

-V Print the version number of the compiler.

-I*directory*

Library files will be searched for in the named directory as well as the directory of the source file and the standard library /usr/lib/Ada. Any number of -I options may be given. Directories are searched in the order (i) directory of source file; (ii) directories specified by -I options (in the order given); (iii) /usr/lib/Ada.

-lx This option is an abbreviation for the library name '/lib/libx.a', where x is a string. If that does not exist, *ld* tries '/usr/lib/libx.a'. A library is searched when its name is encountered, so the placement of a -l is significant.

-Rstring

Suppresses the following run time checks (compare pragma suppress) according to the characters in the given string.

a	access_check
d	discriminant_check
i	index_check
l	length_check
r	range_check
z	division_check
o	overflow_check
e	elaboration_check
s	storage_check
x	all the above checks are suppressed

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

No Linker Options.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

type INTEGER is range -(2**31)..(2**31)-1;
type SHORT_INTEGER is range -(2**15)..(2**15)-1;
type BYTE_INTEGER is range -128..127;
type FLOAT is digits 6 range -3.402_82E+38..3.402_82E+38
type LONG_FLOAT is digits 15 range -1.79769E+308..1.79769E+308;
type DURATION is delta 0.0167 range -16777215.0..16777215.0

end STANDARD;
```

Appendix F: Implementation-dependent characteristics

In accordance with the LRM this section describes all the implementation-dependent characteristics of the York ACE Ada Compiler Release 5.

1. Implementation-dependent pragmas

Two implementation-dependent pragmas are provided - pragma LIBNAME and pragma STACKSIZE.

1.1. Pragma LIBNAME

This pragma is used to associate a subprogram body written in a language other than Ada with a corresponding Ada subprogram specification in a package specification. The other-language body is supplied in lieu of an Ada body. The pragma must be placed immediately before the specification of the subprogram for which the foreign body is to be supplied. The single argument to pragma LIBNAME is the link-symbol of the foreign body. For C this is the routine name (case is significant). This pragma may also apply to objects and exceptions.

e.g.

```
-- Ada
package TEST is
  pragma LIBNAME("iadd");
  procedure IADD(I,J: in out INTEGER);
end TEST;

-- C body for IADD
void iadd(a,b)
int *a,*b;
{
  *a = *a + *b;
}
```

1.2. Pragma STACKSIZE

This pragma can be used by the user to control the amount of storage allocated to any task. Stack space for tasks is taken from the heap. By default, each task is allocated 16K bytes of storage. Pragma *STACKSIZE* may appear at any point within a declarative part. The single argument, a static integer expression, specifies the storage required in bytes.

In the following example *TASK1* will be allocated the default stack size of 16K bytes, *TASK2* and *V1* will both be allocated 32K bytes, *V2* will be allocated 64K bytes.

```
package TEST is

  task TASK1;                    -- STACKSIZE not used

  ONE_K : constant := 1024;

  pragma STACKSIZE(32*ONE_K);   -- first use of STACKSIZE
  task TASK2;
  task type TASKTYPE;
  V1 : TASKTYPE;

  pragma STACKSIZE(64*ONE_K);   -- second use of STACKSIZE
  V2 : TASKTYPE;

end TEST;
```

2. Implementation-dependent attributes

There are no implementation-dependent attributes.

3. Package SYSTEM

```
package SYSTEM is
  type ADDRESS is new integer;
  type NAME is (UNIX);
  SYSTEM_NAME      : constant NAME := UNIX;
  STORAGE_UNIT     : constant := 8;
  MEMORY_SIZE      : constant := 214_748_3647;
  MIN_INT          : constant := -2_147_483_648;
  MAX_INT    : constant := 214_748_3647;
  MAX_DIGITS       : constant := 15;
  MAX_MANTISSA     : constant := 31;
  FINE_DELTA       : constant := 0.000_000_000_465_661_287_307_739_257_8125;
  TICK             : constant := 0.01;
  subtype PRIORITY is INTEGER range 0..4;
end SYSTEM;
```

4. Restrictions on representation clauses

Restrictions on record representation clauses are as follows:

The static simple expression in any alignment clause must be 1, 2 or 4.

Component clauses for fields whose types are non-static or whose types depend upon discriminants are not allowed.

Fields with a type which is not an array or record type must have a bit length which is less than or equal to 32.

Fields with a type which is an array or record type must begin on a storage unit boundary (1 byte or 8 bits) and must have a field length which is a multiple of a storage unit.

Fields with a discrete type may be packed to a field length which is not a multiple of a storage unit (as long as this is feasible for the range of values for the type) but the compiler will refuse to do this for array and record types.

5. Implementation-generated names

Implementation-dependent components are not implemented.

6. Interpretation of expressions in address clauses

Expressions must be of type *INTEGER*.

7. Restrictions on unchecked conversions and unchecked deallocations

There are no restrictions on unchecked conversions and unchecked deallocations.

8. Implementation-dependent characteristics of the input-output packages

The following is a list of features particular to this implementation which relate to the predefined input-output packages:

File names of more than 14 characters (excluding path prefix) cause *USE_ERROR* to be raised.

The actual parameter to "FORM" must be the empty string ("").

Files produced by the *CREATE* procedure with a null string as argument are created in */tmp* with a unique name, and then unlinked.

After the raising of *USE_ERROR*, the file affected is in an undefined state.

The procedure *CREATE* does a *UNIX* create on the file, i.e. it is created if it did not exist, and truncated to the empty file if it did.

The procedure *OPEN* opens a file which must exist. If the file is opened as an OUT_FILE, it is truncated to the empty file.

Attempts to RESET pipes or terminals without changing mode result in *USE_ERROR* being raised. Resetting them with a change of mode has undefined results.

Package *DIRECT_IO* will only work in conjunction with terminals or pipes if they are used sequentially. Attempts to RESET these files raise *USE_ERROR*.

Package *TEXT_IO* produces the following character sequences for the various terminators:

line terminator:	ASCII.LF
page terminator:	line terminator followed by ASCII.FF
file terminator:	page terminator followed by end of file

Package *TEXT_IO* interprets the following character sequences for the various terminators:

line terminator:	ASCII.LF
page terminator:	optional line terminator followed by ASCII.FF or end of file
file terminator:	optional page terminator followed by end of file

For packages *DIRECT_IO* and *TEXT_IO*, type *COUNT* is defined:

```
"type COUNT is range 0 .. 65536;"
```

For package *TEXT_IO*:

```
"subtype FIELD is INTEGER range 0 .. 40;"
```

INPUT-OUTPUT

```
DIRECT_IO.COUNT'LAST  65536
TEXT_IO.COUNT'LAST    65536
FIELD'LAST               40
```

9. Generic units

This release of the compiler allows generic declarations, their bodies and any subunits to be placed in separate files. The only proviso is that the generic body (including any subunits) must either be compiled before instantiations of the generic are made or be in the same file as the generic specification. The compiler will generate error messages and warnings if this is not the case.

In addition, instantiation of the standard library packages *DIRECT_IO* and *SEQUENTIAL_IO* with unconstrained types is not allowed and will result in an error message from the compiler. *TEXT_IO.ENUMERATION_IO* will raise *USE_ERROR* if instantiated with numeric types.

10. Numeric types

10.1. Integer types

Three predefined integer types are implemented as follows.

BYTE_INTEGER

```
BYTE_INTEGER'FIRST          -128
BYTE_INTEGER'LAST           127
BYTE_INTEGER'SIZE           8
```

SHORT_INTEGER

```
SHORT_INTEGER'FIRST      -32_768
SHORT_INTEGER'LAST       32_767
SHORT_INTEGER'SIZE       16
```

INTEGER

```
INTEGER'FIRST      -2_147_483_648
INTEGER'LAST        2_147_483_647
INTEGER'SIZE        32
```

10.2. Floating Point types

Two predefined floating point types are implemented (*FLOAT* and *LONG_FLOAT*) with the following attributes:

	FLOAT	LONG_FLOAT
'DIGITS	6	15
'EMAX	84	204
'EPSILON	9.536_74E-07	8.88178E-16
'FIRST	-3.402_82E+38	-1.79769E+308
'LARGE	1.934_28E+25	2.57110E+61
'LAST	3.402_82E+38	1.79769E+308
'MACHINE_EMAX	128	1024
'MACHINE_EMIN	-125	-1021
'MACHINE_MANTISSA	23	52
'MACHINE_OVERFLOWS	TRUE	TRUE
'MACHINE_RADIX	2	2
'MACHINE_ROUNDS	TRUE	TRUE
'MANTISSA	21	51
'SAFE_EMAX	84	204
'SAFE_LARGE	1.934_28E+25	2.57110E+61
'SAFE_SMALL	2.584_94E-26	1.94469E-62
'SIZE	32	64
'SMALL	2.584_94E-26	1.94469E-62

10.3. Fixed Point types

For any fixed point type, *FIXED*, the following attributes apply:

```
FIXED'SIZE
FIXED'MACHINE_OVERFLOWS    TRUE
FIXED'MACHINE_ROUNDS       FALSE
```

10.4. Type DURATION

The predefined fixed point type *DURATION* is declared as

```
type DURATION is delta 0.0167 range -16777215.0 .. 16777215.0;
```

11. Accuracy of universal and static expressions

Evaluation of static expressions by the compiler is exact. Rational arithmetic is used followed by rounding to the accuracy requested. Universal expressions that are not static are evaluated with the same range and accuracy as the predefined types *LONG_FLOAT* and *INTEGER*.

11.1. Miscellaneous restrictions

The compiler was written to impose as few restrictions on user programs as possible. The following points should however be noted.

There is no limit placed on the number of characters per line.

A maximum of 128 characters per identifier is allowed. If the length of an identifier exceeds this limit it will be truncated by the compiler and an error message will be issued.

A maximum of 256 characters per numeric literal is allowed. If the length of a numeric literal exceeds this limit it will be truncated by the compiler and an error message will be issued.

Temporary files do not have names in this implementation.

The main program should be a parameterless procedure otherwise the effect is undefined.